The Bell System Technical Journal
Vol. 52, No. 3, March, 1973
Printed in U.S.A.

Dynamic Data Reallocation in Bubble Memories

By P. I. BONYHARD and T. J. NELSON

(Manuscript received October 10, 1972)

Bubble technology offers several operations that have no equivalents in technologies based on magnetic recording. Examples of such operations are: transfer, reversal of the direction of propagation, and opening and closing of gaps in the data stream. This paper shows how such operations can be used to dynamically reallocate data in the bubble memory, causing it to become an integrated memory hierarchy. A considerable improvement in performance results. A model is presented which relates the bubble memory with dynamic reallocation to stack processing, a technique used in the evaluation of memory hierarchies. With the aid of this model it becomes possible to calculate the performance of the bubble memory using published data derived from the traces of selected typical programs. Memory design is optimized for the execution of such programs. Design parameters are proposed for a 2-Mb bubble memory with 128 detectors which, in the execution of the type of program for which data were available, requires an average of only 8.8 shifts for access and an average of 12.1 shifts per memory cycle. If bubbles are propagated at a rate of 1 MHz, the average access and cycle times for this memory become 8.8 μs and 12.1 μs, respectively. Such performance, in conjunction with the low cost per bit offered by bubble technology, is expected to have a major impact. The performance of this memory, when operated in conjunction with a faster buffer, is also calculated. The use of a 64-kb buffer is shown to reduce the average number of shifts for access to 1.05, and the average number of shifts per cycle to 1.9.*

I. INTRODUCTION

One major area of application of magnetic bubble technology is mass memory.[1] The potential advantages of bubbles over disk files and drums are: shorter access time, lower cost and power dissipation,

* The contents of this paper were presented at the 1972 Intermag Conference by P. I. Bonyhard.

reduced volume, and a higher degree of modularity.[2] Moreover, there are several operations which can be performed on the information in a bubble memory which cannot be performed by conventional magnetic recording. The purpose of this paper is to show that a magnetic bubble memory can function as a hierarchy, given certain operations which are not difficult to realize.

The operations relevant to this paper are: transfer, instantaneous reversal of the direction of propagation, and removal of bubbles from a propagate channel while closing the gap left by them or creation of a gap while inserting new bubble information into the channel. Transfer of bubbles from one propagate channel to another already plays an important role in the design of bubble mass memories.[1,2] By a process of (*i*) propagating forward n cycles, (*ii*) removing bits from the channel and closing the gap, (*iii*) propagating backward n cycles, and (*iv*) opening a gap and reinserting the bubbles, a permutation of the information in the channel is effected. It will be shown how a simple algorithm using one permutation element per propagate channel causes the memory to function as an integrated hierarchy. We also show how this dynamic reallocation of data can be combined with the major-minor loop organization[1] so as to optimize memory cost performance. It is concluded that the improvement is so substantial as to have a major impact in the computer industry.

II. DYNAMIC DATA REALLOCATION IN CLOSED LOOP SHIFT REGISTERS

Consider an assembly of simple, closed loop shift registers as shown in Fig. 1a. Information propagates in all registers synchronously under the influence of a common rotating drive field. One page of information is stored along a horizontal line, a particular page being formed by the black dots in Fig. 1a. It is inherent in the dynamic scheme that each page must carry its own reference address and some of the bits of the page are devoted to this purpose. Thus, extra loops, totaling $\log_2$ (number of pages), have to be provided.

Upon request for a specified page, information is shifted clockwise until the appropriate group of detectors, the position of which is marked D in Fig. 1b, detects the right address. This takes, say, x cycles of propagation. Now the page can be read or rewritten. The memory next is reset by shifting x cycles counterclockwise. Using the circuit arrangement of Fig. 1b, the addressed page will remain arrested at the detectors while all other pages move back. A T-bar realization of Fig. 1b is shown in Fig. 2. This design has been operated quasi-statically to demonstrate feasibility.

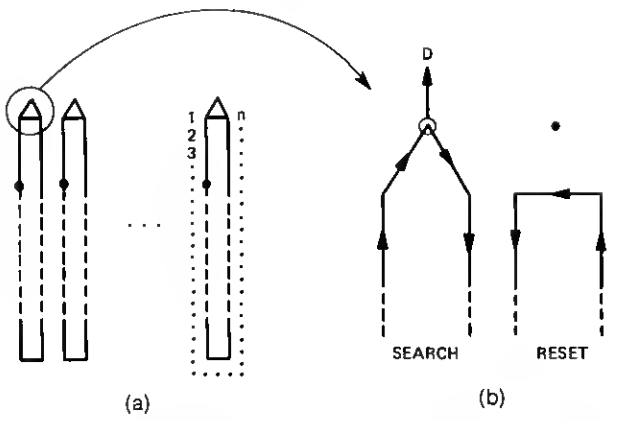

Fig. 1—Assembly of shift register loops for dynamic address reallocation.

Now let the physical page locations be numbered 1, 2, 3, $\cdots$, n, according to their distance from the detectors. A given page will reside in a location, the number of which is equal to the number of requests

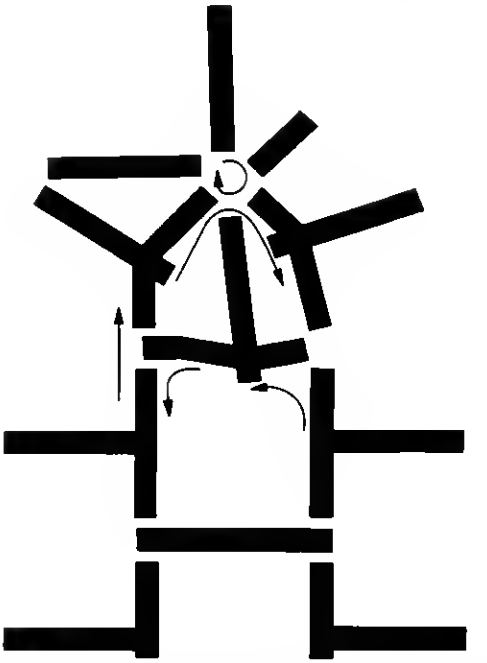

Fig. 2—Magnetic circuit used to realize the function of Fig. 1b.

that have been made, since the page was last requested, for pages originally in locations with higher numbers. Pages never requested will lie in the locations with the highest numbers. Thus recently used pages are near the "top," whereas less recently used pages are farther down. This replacement algorithm has been discussed before in the literature and has been named "the stack."[3]

It will now be shown how the average number of shifts necessary to reach an addressed page in the bubble stack can be calculated. Consider the stack to be arbitrarily divided into two parts, one part consisting of pages in locations 1, 2, $\cdots$, k_i, and the other part of pages in locations $k_i + 1$, $k_i + 2$, $\cdots$, n. It should be recognized that the first k_i page locations can be thought of as forming a "buffer" and

TABLE I—HIT RATIO DATA (REPRODUCED FROM REF. 4)

Buffer Size (Bits)	Classes	Page Size (Bits) 128	256	512	1k	2k	4k	8k	16k*	32k*
8k	1	0.894	0.915	0.928	0.924	0.884	0.792	0.495	—	—
	4	0.895	0.916	0.921	0.904	0.791	—	—	—	—
	16	0.891	0.903	0.860	—	—	—	—	—	—
	64	0.857	—	—	—	—	—	—	—	—
16k	1	0.931	0.949	0.957	0.958	0.950	0.912	0.824	0.56	—
	4	0.931	0.948	0.954	0.955	0.933	0.808	—	—	—
	16	0.930	0.943	0.943	0.909	—	—	—	—	—
	64	0.921	0.913	—	—	—	—	—	—	—
32k	1	0.951	0.969	0.973	0.978	0.977	0.966	0.939	0.86	0.64
	4	0.955	0.969	0.973	0.977	0.974	0.951	0.834	—	—
	16	0.955	0.968	0.972	0.970	0.933	—	—	—	—
	64	0.955	0.963	0.948	—	—	—	—	—	—
	256	0.934	—	—	—	—	—	—	—	—
64k	1	0.977	0.986	0.988	0.985	0.987	0.987	0.984	—	—
	4	0.981	0.986	0.988	0.986	0.987	0.985	0.965	—	—
	16	0.981	0.985	0.988	0.987	0.983	0.954	—	—	—
	64	0.979	0.984	0.985	0.974	—	—	—	—	—
	256	0.974	0.971	—	—	—	—	—	—	—
128k	1	0.985	0.993	0.994	0.996	0.993	0.992	0.994	—	—
	4	0.990	0.993	0.994	0.996	0.994	0.992	0.993	—	—
	16	0.990	0.994	0.995	0.997	0.995	0.991	0.957	—	—
	64	0.990	0.994	0.995	0.995	0.985	—	—	—	—
	256	0.989	0.992	0.986	—	—	—	—	—	—
256k	1	0.989	0.996	0.997	0.997	0.999	0.994	0.997	—	—
	16	0.994	0.996	0.997	0.998	0.998	0.996	0.997	—	—
	32	0.994	0.996	0.998	0.998	0.998	0.997	0.997	—	—
	64	0.994	0.996	0.998	0.998	0.998	0.988	—	—	—
	256	0.994	0.996	0.997	0.992	—	—	—	—	—

the last $n - k_i$ pages a "memory" in the terminology conventionally used for two-level memory hierarchies.[3,4] Whenever a page is brought from the memory to the buffer, the page currently in the k_ith position moves from the buffer to the memory. Clearly, this page is the least recently used page in the buffer, so that the replacement algorithm in this two-level hierarchy is "least recently used" (LRU). Hit ratios, that is, fractions of all requests that can be satisfied from the buffer without reference to the memory, can be found in literature. A particularly useful set of hit ratio data is given in Table II of Ref. 4 and is reproduced as Table I of this paper. The columns marked with asterisks contain entries obtained by graphical extrapolation from the original data.

In a two-level hierarchy, derived by cutting the huhhle stack at the k_ith location, the number of bits per page is equal to the number of bubble loops, say, ℓ. The numher of hits in the huffer is then ℓk_i. If the corresponding hit ratio is h_i, then the average number of shifts necessary per request might be estimated as

$$\bar{S} = \frac{k_i - 1}{2} h_i + \frac{n + k_i - 1}{2} (1 - h_i).$$

This is the average number of shifts in the huffer times the prohahility of a hit, plus the average numher of shifts to the memory times the probability of a miss.

The bubble stack, however, can be cut at any location. A better estimate is obtained by dividing the stack into $m > 2$ levels

$$\bar{S} = \sum_{i=1}^{m} \frac{k_i + k_{i-1} - 1}{2} (h_i - h_{i-1}) \tag{1}$$

where $k_o = 0$, $k_m = n$, and $h_m = 1$. The best approximation to $\bar{S}$ is achieved when every value of k hetween 1 and n is taken into account:

$$\bar{S} \to \sum_{k=1}^{n} (k - 1)(h_k - h_{k-1}).$$

This result is, of course, self-evident, as $k - 1$ is the distance of the kth location and $h_k - h_{k-1}$ is the probability of hitting the kth location.

$\bar{S}$ has been calculated on the hasis of eq. (1) using the entries that can be found in Table I. It is an overestimate hecause it divides the prohability equally among the levels hetween entries. Actually, the

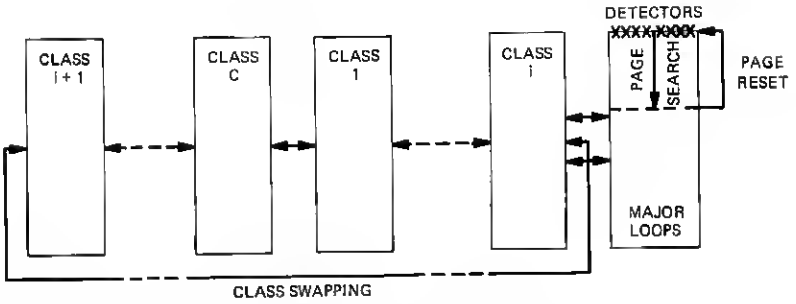

Fig. 3—Schematic representation of the operation of a major-minor loop organized memory with dynamic data reallocation in the major loops.

probability must be monotonically decreasing. The results are tabulated below:

Number of loops	128	256	512	1K	2K	4K	8K
Minimum number of bits per loop	2K	1K	512	256	128	64	32
Average number of shifts	57.8	25.9	12.13	5.67	3.08	1.44	1.05

The minimum number of bits per loop listed in the above tabulation is based on the information kindly provided by tbe author of Ref. 4 to the effect that the average program size used in deriving the hit ratio data was 256 kb. Thus closed loop shift registers can access data at least ten times faster with dynamic reallocation. A magnetic bubble memory with this feature is, in fact, an integrated hierarchy.

III. DYNAMIC DATA REALLOCATION IN MAJOR-MINOR LOOP ORGANIZED MEMORY PLANES

Dynamic data reallocation in closed loop shift registers does not appear to be a very attractive bubble mass memory design. Each loop must have its own detector, so that either the number of detectors or the average number of shifts is high. The average number of shifts for a given number of bits per detector does not compare too favorably with tbe major-minor loop organization.[1] Dynamic reallocation may, however, be combined with major-minor loop organization to form an extremely attractive design. Dynamic reallocation can be performed in the major loops only, in the minor loops only, or in botb. The last of these three options is probably too complex to be of practical use. A design based on tbe first option is presented below. The second option

may he a valid alternative hut, at least in the authors' opinion, it is less attractive.

Figure 3 illustrates schematically how the system operates. The information that may occupy the assemhly of all the major loops concurrently forms a class. One of the classes is selected hy shifting information in the minor loops until the required class is aligned for transfer, and then transferring it into the major loops. Two hits from each minor loop are transferred and the major loop is filled completely. Consequently, two bits of each class are stored in each minor loop in each memory plane. A plane consists of one major loop and all the minor loops associated with it, as well as one detector and one controlled huhhle generator operating on the major loop. Once the right class resides in the major loops, the major loops operate exactly as the assemhly of loops described in the previous section. The net result is that page locations are dynamically reallocated within each class, hut pages are never permitted to cross class houndaries. Hit ratios that correspond to such a multiclass system are given in Ref. 4, and also appear in Tahle I.

It should he recognized that a page in this system consists of all those hits that may concurrently he detected. Thus there is one hit of each page in each plane. Also, the transfer mechanism considered here is of the "conductor" variety,[1] so that reversing the sense of propagation may he freely used to accomplish dynamic address reallocation in the major loops. As this consists of an equal numher of forward and reverse shifts, the gaps left in the minor loops are correctly aligned at the time the class is to he transferred hack.

Reaching a page in the memory is accomplished in two steps. First the right class is positioned and transferred into the major loops, and then the page is hrought to the read/write port. The latter step has already heen discussed in the previous section, hut the former one, class swapping, needs some elahoration. Following the approach taken in Ref. 4, it is assumed that several programs are concurrently resident in the memory. More specifically, the memory size is chosen to he 2 Mh, whereas the average program size is 256 kh, so that an average of 8 programs may share the memory. It is assumed that each program resides in some numher of contiguous classes which will he called active classes for the program currently heing executed. The class occupying the major loops at any given time can he looked upon as a single-page huffer of a single-class two-level hierarchy, with all the other classes forming the rest of the pages in the memory. Of course,

for a single-page buffer the replacement algorithm is trivial. However, as can be seen in Table I, the hit ratios associated with such single-page buffers are still fairly high. Consequently, there is a good chance that the next request is addressed to the class already in the major loops, provided that the addressed class is not transferred back to the minor loops until it is known which class is addressed next. Therefore, it makes good sense to operate in this manner.

The average number of shifts necessary to bring out the new class, provided that a "class-page" failure has occurred, will now be calculated. It is assumed that the particular program executed at the time resides in m contiguous classes and the ith class was addressed last. In the absence of any further information, it appears to be reasonable to assume that each other class is equally likely to be addressed next. Consequently, the average number of shifts is

$$\bar{S}_m = \frac{2}{m}\left(\sum_{j=1}^{i-1} j + \sum_{j=1}^{m-i} j\right) = \frac{(i-1)i}{m} + \frac{(m-i+1)(m-i)}{m}.$$

It has to be remembered that there are two bits per class in each minor loop. The average $\bar{S}_m$ must be further averaged as i assumes all values between 1 and m, so that

$$\bar{\bar{S}}_m = \frac{1}{m}\sum_{i=1}^{m} \bar{S}_m = \frac{2}{m^2}\sum_{i=1}^{m} (i-1)i = \frac{2}{3}\frac{m^2-1}{m}.$$

To this two further shifts must be added to accomplish in and out transfer of two bits per minor loop. The total average number of class swapping shifts per page request is

$$S_{cs} = (1 - h_{cs})\left(\frac{2}{3}\frac{m^2-1}{m} + 2\right) \tag{2}$$

where h_{cs} is the class hit ratio discussed earlier in this section, and is the number of active classes.

To the class swapping shifts, given by eq. (2), one must add the average number of shifts necessary to reach a page within the selected class to get the total average number of access shifts per request to the memory. The average number of shifts within the selected class is given by eq. (1) with the hit ratios taken for the appropriate number of classes. The latter quantity must be doubled when calculating the

average number of shifts per memory cycle. The results are given below for three alternative designs labeled A, B, and C.

	A	B	C
Bits per memory	2M	2M	2M
Planes per memory	128	128	128
Bits per plane	16k	16k	16k
Classes	64	128	256
Bits per minor loop	128	256	512
Minor loops per plane	128	64	32
Bits per major loop	256	128	64
Active classes	8	16	32
Active bits per minor loop	16	32	64
Class hit ratio	≈0.64	≈0.56	0.495
Class swapping shifts	7.25	12.63	23.3
Class swapping shifts per request	2.61	5.56	11.8
Page search shifts	7.00	3.26	1.40
Total shifts per access	9.61	8.82	13.2
Total shifts per cycle	16.61	12.08	14.6

Details of the page search shift calculations are given in Table II. The memory size was chosen to be consistent with Ref. 4, where the

TABLE II—CALCULATION OF THE AVERAGE NUMBER OF PAGE SEARCH SHIFTS FOR THREE DESIGNS

Design	k_i	$\frac{k_i + k_{i-1} - 1}{2}$	h_i	$h_i - h_{i-1}$	$\frac{1}{2}(h_i - h_{i-1}) \times (k_i + k_{i-1} - 1)$
A	8	3.5	0.894	0.894	3.13
	16	11.5	0.931	0.037	0.43
	32	23.5	0.955	0.024	0.56
	64	47.5	0.981	0.026	1.24
	128	95.5	0.990	0.009	0.86
	256	191.5	0.994	0.004	0.77
					6.99
B	4	1.5	0.891	0.891	1.34
	8	5.5	0.930	0.039	0.21
	16	11.5	0.955	0.025	0.29
	32	23.5	0.981	0.026	0.61
	64	47.5	0.990	0.009	0.43
	128	95.5	0.994	0.004	0.38
					3.26
C	2	0.5	0.880	0.880	0.440
	4	2.5	0.934	0.054	0.135
	8	5.5	0.955	0.021	0.115
	16	11.5	0.980	0.025	0.287
	32	23.5	0.990	0.010	0.235
	64	47.5	0.994	0.004	0.190
					1.402

data came from, and the plane size was chosen as the one that would provide an economical design on the hasis of current cost estimates.

Each of the three designs has an overhead associated with it, hecause of the extra planes needed to store the address tags of the reallocated pages. This overhead for designs A, B, and C is 6.3 percent, 5.5 percent, and 4.7 percent, respectively.

IV. THE USE OF A BUFFER TO IMPROVE PERFORMANCE

The hest design outlined in the preceding section gave an average cycle time of ahout 12.1 shifts or, assuming 1-MHz operation, 12.1 μs. The question arises of how this figure may he further improved hy the use of a smaller suhmicrosecond cycle time core or integrated circuit huffer. The answer can he readily calculated, provided that the huffer is divided into as many classes as there are active classes in the memory, and that the page size in the huffer is the same as in the memory. Now, if there are k_B page frames per class in the huffer, then the pages currently residing in location 1, 2, $\cdots$, k_B of all active classes will also appear in the huffer. This statement is not quite correct if the huffer stores only read (fetch) data, whereas write (store) data are sent from the central processor directly to the memory, hut the difference is prohahly negligihle.

It is, of course, also necessary to uniquely assign all nonactive memory classes to buffer classes, so that program swapping can take place. This can he done as follows. In terms of Design B, let the 7 most significant hits of the 14-hit page address he the class address. As each program occupies contiguous classes, the 4 least significant hits of the class address refer to classes in the same program for a typical program size. Thus the 1st, 2nd, 3rd, and 4th least-significant hits of the memory page address should he considered to he the huffer class address.

The average numher of shifts necessary in the memory per request can now he calculated as follows. If the huffer hit ratio is h_B, then h_{cs} in eq. (2) should be replaced hy h_B, whereas all contrihutions to the page search shifts, calculated in Tahle II, hy values of $k_i \leq k_B$ should he neglected. The results are tabulated below, still for Design B.

Buffer size (bits)	8k	16k	32k	64k
Buffer bit ratio	0.891	0.930	0.955	0.981
Pages per class in buffer	4	8	16	32
Class swapping shifts	1.37	0.88	0.56	0.24
Page search shifts	1.92	1.71	1.42	0.81
Total access shifts	3.29	2.59	1.98	1.05
Total shifts per cycle	5.21	4.30	3.40	1.86

V. DISCUSSION AND CONCLUSION

The availability of memories costing approximately the same or less per bit as a disk file or drum with cycle times of about 10 μs is likely to have a major impact on the computer industry. This appears to he the answer to the system designer's old dream of a memory at core speeds and disk costs. It looks like we can fulfill this dream with buhble memories, provided that we can shift bubbles at 1-MHz rates, which appears to be a reasonable objective. It may he added that thc organizations discussed here should be also applicahle to other forms of scrial mcmory technologies, MOS registers, charge coupled registers, etc.

The major applications for bubhle memories with dynamic address rcallocation are likely to he in mini- and midi-computers and in other systems with relatively less powerful central processors. The larger systems and those with faster processors will probably find the buffered configuration more attractive. For instance, one may use a 64-kh, 0.5-μs access time, 1-μs cycle time buffer to give an average access time of about 1.5 μs, and an average cycle time of about 3.0 μs for the 2-Mb bubble memory. In many systems this would be vastly preferable to an all-integrated-circuit memory costing much more for the same capacity, even if the latter memory has a cycle time of 0.1 μs.

Further work should be directed towards the assessment of the applicability of these techniques to electronic telephone switching systems.

VI. ACKNOWLEDGMENTS

The authors havc benefited from helpful discussions with many individuals: A. D. Friedman, J. E. Geusic, B. W. Kernighan, M. D. McIlroy, P. R. Menon, P. C. Michaelis, and H. E. D. Scovil in particular. Also, we thank R. L. Mattson who provided a preprint of the work which inspired this paper.

REFERENCES

1. Bonyhard, P. I., et al., "Applications of Bubble Devices," IEEE Trans. Magnetics, *MAG-6*, September 1970, pp. 447–451.
2. Bobeck, A. H., "Magnetic Domain Devices–A Tutorial," presented at the 1971 Intermag Conf. (also to be published in Scieutific American).
3. Mattson, R. L., et al., "Evaluation Techniques for Storage Hierarchies," IBM Syst. J., *9*, No. 2, 1970, pp. 78–117.
4. Mattson, R. L., "Evaluation of Multilevel Memories," IEEE Trans. Magnetics, *MAG-7*, December 1971, pp. 814–819.